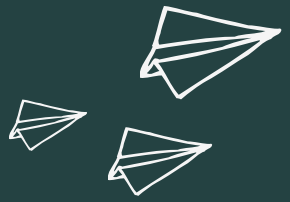


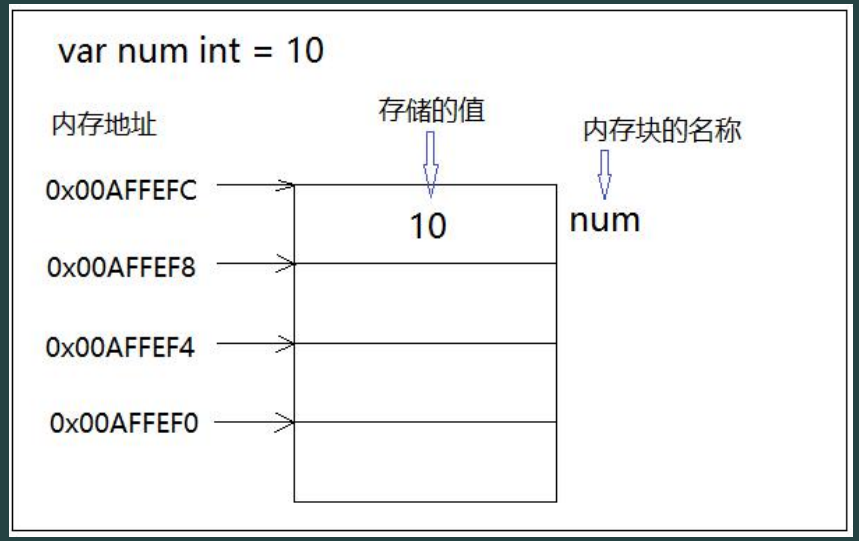
go 语 言 编 程

目录页

第二章
Part two

2.1 变量

在计算机程序中，变量就是对一块数据存储空间的命名，变量的值就是存储的数据。在程序的运行期间，变量的值是可以改变的。



2.1.1 变量命名

Go语言中的变量、常量、类型、函数和包等所有的命名和C语言一样都遵循这样一个简单的命名规则:

一个名字必须以一个字母或下划线开头，后面可以跟任意数量的字母、数字或下划线。对于字母区分大小写，例如:name和Name是两个不同的名字。

2.1.1 变量命名

命名不能与关键字相同，Go语言提供了25个关键字，只能在特定的语法中使用。

25 个关键字				
break	default	func	interface	select
case	defer	go	map	struct
chan	else	goto	package	switch
const	fallthrough	if	range	type
continue	for	import	return	var

2.1.1 变量命名

此外，Go语言还有大约30多个预定义的名字，主要用于内建的常量、类型和函数。这些名字不是关键字，虽然可以重新定义和使用，但建议尽量不要重新定义，以免造成语义混乱问题。

内建常量	true false iota nil
内建类型	int int8 int16 int32 int64
	uint uint8 uint16 uint32 uint64 uintptr
	float32 float64 complex128 complex64
	bool byte rune string error
内建函数	make len cap new append copy close delete
	complex real imag
	panic recover

2.1.2 变量声明

1) 一般声明格式

var 变量名 类型 = 表达式

2) 简短变量声明

名字 := 表达式

3) 多个变量声明

```
var i, j, k int = 1, 2, 3
var m, n int
var a, b, c = 1, 2, 3
d, e, f := 1, 2, 3
name, age := "张三", 20
```

```
var (
        name string
        age  int
    )
```

2.1.3 变量赋值

1) 简单赋值

```
var i int
i = 1       //简单赋值
```

2) 复合赋值运算符

```
var i int
i = i + 1
i += 1     //与 i = i + 1 等价
```

3) 多重赋值

```
x, y=y, x
```

4) _标识符

```
_, err := io.Copy(dst, src)//丢弃字节数
```

2.2 常量

在Go语言中，常量是指编译期间就已知且不可改变的值。

1) 使用const声明常量

一个常量的声明可以使用限定类型，但不是必须的。如果没有显示指定类型，那么它是无类型常量。

常量的右值也可以是常量表达式

如果是批量声明常量，除第一个之外的其它常量的右值都可以省略，默认使用前面常量的初始化表达式。

2.2 常量

2) iota常量生成器

Go语言中预定义常量有：true，false和iota。iota用于生成一组相似规则的初始化的常量。在一个const声明语句中，在第一个声明常量的所在行，iota会被设置为0，然后每一个有常量的行加1。这种定义方法在Go语言中通常用于定义枚举值。

2.3 数据类型

Go语言将数据类型分为四类：基础类型、复合类型、引用类型和接口类型。

1.基本数据类型：数值、字符串和布尔型。

2.复合数据类型：数组和结构体。

3.引用类型：指针、切片、字典、函数和通道。

4.接口类型。

2.3.1 整数

1> 整数类型

Go语言的数值类型包含了几种不同长度的整数、浮点数和复数。每种数值类型都决定了对应的取值范围和是否支持正负号。

类型	长度(字节)	取值范围
int8	1	(0～255)
uint8	1	(-128～127)
int16	2	(0～65535)
uint16	2	(-32768～32767)
int32	4	(-2147483648～2147483647)
uint32	4	(0～4294967295)
int64	8	(-9223372036854775808～9223372036854775807)
uint64	8	(0～18446744073709551615)
int	4 或 8	与机器字长和编译器都有关系
uint	4 或 8	与机器字长和编译器都有关系
uintptr	4 或 8	32 平台 4 个字节，64 位平台 8 个字节，底层编程才需要
byte	1	与 uint8 等价，通常表示一个 unicode 字符编码
rune	4	与 int32 等价，一般强调是一个原始数据而不是一个小整数。在一个字符串中，表示一个字符对应 utf8 的码点。

2>运算符

Go语言提供了丰富的内置运算符，包括算术运算符、比较运算符、逻辑运算符、位运算符、赋值运算符和其它运算符等。

算术运算符

运算符	描述
+	加
-	减
*	乘
/	除
%	模运算(求余数)
++	自增
--	自减

第二章 变量、常量和数据类型

关系(比较)运算符：

运算符	描述
==	相等
!=	不等
<	小于
<=	小于或等于
>	大于
>=	大于或等于

逻辑运算符：

运算符	描述
!	非
&&	与
||	或

第二章 变量、常量和数据类型

位运算符：

运算符	描述
&	位与 and（左侧和右侧都为 1，则为 1；否则为 0）
|	位或 or(左侧或右侧只要有一个为 1，结果为 1；都为 0 结果才为 0)
^	位异或 xor（相同为 0，不同为 1）
&^	位清空 and not(右侧是 0，左侧数不变;右侧是 1，则左侧数清零)
<<	左移
>>	右移

X=2,y=15	二进制结果	十进制结果
0000 0010 & 0000 1111	0000 0010	2
0000 0010 | 0000 1111	0000 1111	15
0000 0010 ^ 0000 1111	0000 1101	13
0000 0010 &^ 0000 1111	0000 0000	0
0000 0010<<3	0001 0000	16
0000 0010>>1	0000 0001	1

第二章 变量、常量和数据类型

运算符优先级：

分类	描述	关联性
后缀	() [] -> . ++ - -	左到右
一元	+ - ! ~ ++ - - (type)* & sizeof	右到左
乘法	* / %	左到右
加法	+ -	左到右
移位	<< >>	左到右
关系	< <= > >=	左到右
相等	== !=	左到右
按位AND	&	左到右
按位XOR	^	左到右
按位OR	|	左到右
逻辑AND	&&	左到右
逻辑OR	| |	左到右
条件	?:	右到左
分配	= += -= *= /= %=>>= <<= &= ^= | =	右到左
逗号	,	左到右

2.3.2 浮点数

浮点数用于表示包含小数点的数据。Go语言提供了两种精度的浮点数，float32和float64。float32与float64之间需要强制转换。强制转换的方式T(V)，T为要转换的目标类型，V需要转换的变量。

1> 浮点数表示

```
var f1 float32
f1 = 10
f2 := 12.0 //带小数点的自动推导为 float64
f2 = float64(f1) //需强制转换
```

2.3.2 浮点数

2>浮点数比较

浮点数不是一种精确的表达方式，所以不能像整型那样直接用==比较。

```
import  "math"
func IsEqual(f1, f2, p float64) bool {
        return math.Abs(f1-f2) < p
}
```

3>科学计数法

把一个数表示成a（1≤a<10，n为整数）与10的幂相乘的形式，这种记数法叫做科学记数法。计算器或计算机表达10的幂是一般是用E或e。

```
f1 := 1.99e+3    //1990
f2 := 1.99e-3    //0.00199
```

2.3.3 复数

Go语言提供了两种精度的复数类型：complex64和complex128，分别对应float32和float64两种浮点数精度。

2.3.4 布尔类型

一个布尔类型的值只有两种:true和false。布尔值不会隐式转换为数值0或1。布尔值可以和&&、||操作符结合，并且可能会有短路行为。

```
var s string
//s = "mazhiguo"
if s != "" && s[0] == 'm' {
	fmt.Println("OK")
} else {
	fmt.Println("error")
}
```

2.3.5 字符串

在Go语言中字符串也是一种基本类型。一个字符串是一个不可改变的字节序列。

1> 字符串常用操作

运算	含义	备注
s1+s2	字符串连接	
len(s)	字符串长度	字符串中的字节数,不是字符数
s[i]	取字符	索引 i 不能越界
s[i:j]	取子字符串	左闭右开，包含 s[i],不包含 s[j]。子字符串是一个新的字符串。 i,j 都可能被忽略，忽略时，从 0 开始，最后一个字符结束。

2> 字符串值不可变

一个字符串包含的字节序列永远不会被改变，当然我们可以给一个字符串变量分配一个新字符串值。

3> 字符串遍历

- 以字节的方式遍历
- 以字符的方式遍历

第二章 变量、常量和数据类型

4> 转义序列

\a	响铃
\b	退格
\f	换页
\n	换行
\r	回车
\t	水平制表符
\v	垂直制表符
\'	单引号
\"	双引号
\\	反斜杠

5> 原生字符串字面值

原生的字符串字面值，用` `代替双引号。可用于编写正则表达式。常用于HTML模板、JSON面值、命令提示信息以及需要扩展到多行的场景。

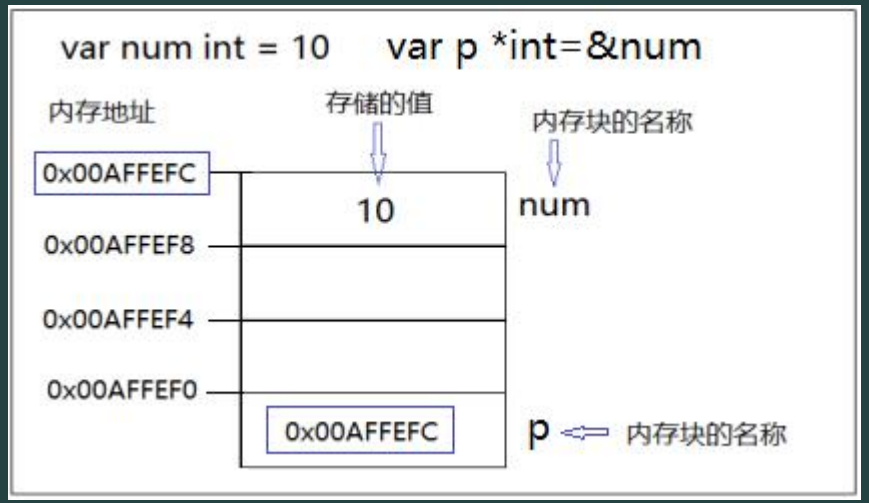
6> unicode和utf8编码

unicode是一种字符串编码。utf8以及utf16是如何以字节的方式存储这个编码，utf8使用1到4个字节表示一个字符。ASCII部分字符只使用1个字节，常用字符部分使用2或3个字节。

字符串可比较，可遍历，但是不可修改。

2.4 指针

指针是一种类型，指针类型的变量称为指针变量。它不同于一般的变量，一般变量存放的是数据本身，而指针变量存放的是数据的地址。



第二章 变量、常量和数据类型

2.4.1声明指针变量

声明指针变量的一般格式如下：var 变量名 *类型

2.4.2 指针操作注意事项：

1.默认值 nil，没有 NULL 常量。

2.操作符 "&" 取变量地址， "*" 通过过指针访问目标对象。

3.不支持指针运算，不支持 "->" 运算符，直接用 "." 访问目标成员

4.不能对指针做加减法等运算

5.不存在函数的指针

2.4.3 数组指针和指针数组

数组指针是指一个指针变量保存的是数组的地址。指针数组是指数组的每个元素都是指针类型。

2.4.4 二级指针(多级指针)

二级指针保存一级指针变量的地址。

2.5 type定义类型

在任何程序中都会存在一些变量有着相同的内部结构，但是却表示完全不同的概念。 Go语言可以通过一个类型声明语句创建了一个新的类型名称，和现有类型具有相同的底层结构。

定义的一般格式： type 类型名字 底层类型

类型声明语句一般出现在包一级，因此如果新创建的类型名字首字母大写，则在包外可以使用。否则，只能在包内使用。

2.6 变量作用域

变量的作用域与语法块相关。

语法块就是一对大括号括住的声明和语句，它决定了内部声明的名字的作用域范围。作用域是一个编译时属性。不要将作用域和生命周期混为一谈，生命周期是指程序运行时变量存在的有效时间段，是运行时的概念。

2.6 变量作用域

全局作用域：对于内置的类型、函数和常量，例如int、len和true等。
包级作用域：函数外声明的名字可以在包的任何源文件中访问。
文件级的作用域：对于导入的包，则是对应文件级的作用域。
函数作用域：函数的参数，返回值以及函数内声明的名字都属于函数的作用域。控制流标号，就是break、continue或goto语句后跟着的那种标号，也是函数级作用域。
另外，每个for、if和switch语句的语法块；每个switch或select的分支也有独立的作用域；当然也有显示书写的语法块(花括号包含的语句)。

2.6 变量作用域

当编译器遇到一个名字引用时，如果它是一个声明，首先从最内层的作用域向全局作用域查找。如果查找失败，则错误。如果名字在内部和外部分别声明过，则内部块的声明首先被找到，它会屏蔽外部同名的声明。

2.7 数据输入输出

1 标准输出函数：Print( )、Println( )和Printf( )

Print( )函数采用默认格式将其参数格式化并写入标准输出。如果两个相邻的参数都不是字符串，会在它们的输出之间添加空格。返回写入的字节数和遇到的任何错误。

```
func Print(a ...interface{}) (n int, err error)
```

Println( )与Print( )函数的功能基本一致，唯一不同的是在输出结束后，自动增加换行。

```
func Println(a ...interface{}) (n int, err error)
```

2.7 数据输入输出

Printf()函数根据format参数生成格式化的字符串并写入标准输出。返回写入的字节数和遇到的任何错误。对应类型的格式化表示如下：

整型，%d；浮点型，%f；字符，%c；字符串，%s；布尔值，%t；值，%v;

```
func Printf(format string, a ...interface{}) (n int, err error)
```

2 标准输入函数：Scan( )、Scanln( )和Scanf( )

Scan( )函数从标准输入扫描文本，将成功读取的空白分隔的值保存进成功传递给本函数的参数。换行视为空白。返回成功扫描的条目个数和遇到的任何错误。如果读取的条目比提供的参数少，会返回一个错误报告原因。

```
func Scan(a ...interface{}) (n int, err error)
```

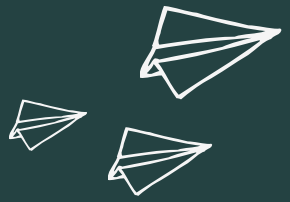
2.7 数据输入输出

Scanln类似Scan，但会在换行时停止扫描。最后一个条目后必须有换行或者到达结束位置。

```
func Scanln(a ...interface{}) (n int, err error)
```

Scanf从标准输入扫描文本，根据format 参数指定的格式将成功读取的空白分隔的值保存进成功传递给本函数的参数。返回成功扫描的条目个数和遇到的任何错误。

```
func Scanf(format string, a ...interface{}) (n int, err error)
```



THANKS

感谢聆听，期待反馈